主要内容

四、网络安全协议（续）

- 传输层安全协议 SSL/TLS
- 应用层安全协议
  - HTTPS
  - SSH（选）
  - DNS安全协议DNSSEC

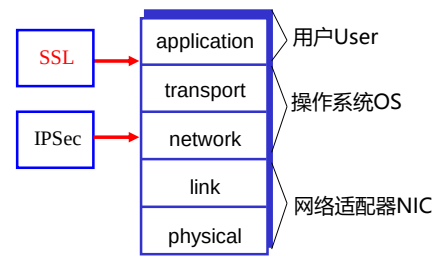
传输层安全 -SSL/TLS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

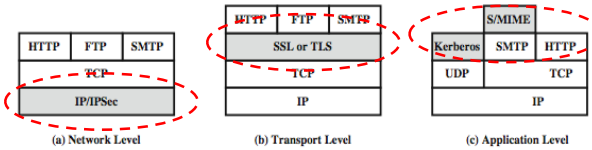
Web 安全需求

- WWW的广泛应用
  - 客户/服务器应用
  - 双向通信；门户和信息出口；软件复杂性；用户的多样性
- Web的安全威胁
  - 完整性 integrity: 木马，消息篡改等
  - 保密性 confidentiality：窃听
  - 拒绝服务 denial of service：DDOS, DOS攻击
  - 认证 authentication：假冒合法用户
- 需要增加安全机制
  - Web服务器
  - 浏览器
  - 浏览器和服务器之间的网络通信

IPSec 与 SSL



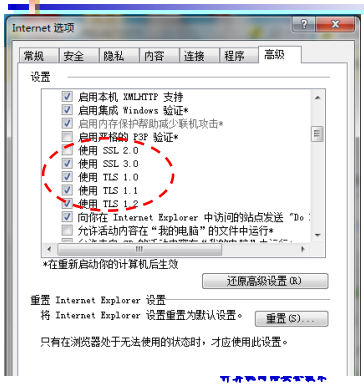
Web安全



SSL/TLS协议历史

- 1994年Netscape开发了安全套接字层SSL(Secure Socket Layer)协议，Web安全机制，提供鉴别与保密服务，SSL 1.0，不成熟，未发布
- 版本和历史
  - 1995年，SSL 2.0，基本上解决了Web通信的安全问题
    - Microsoft公司发布了PCT(Private Communication Technology)，并在IE中支持
  - 1996年发布SSL3.0，增加了一些算法，修改了一些缺陷
  - 1999年，IETF(www.ietf.org)将SSL作了标准化，即RFC2246,并将其称为TLS(Transport Layer Security)，TLS 1.0 也被称为SSL 3.1
  - 2006年和2008年，TLS 1.1 RFC 4346，TLS1.2
  - 2011年，TLS1.2修订版
- 目前，主流浏览器都已经实现了TLS 1.2的支持。
  - TLS 1.0通常被标示为SSL 3.1，TLS 1.1为SSL 3.2，TLS 1.2为SSL 3.3

例：SSL/TLS协议在浏览器中的设置



Internet Explorer 11

SSL协议

- SSL (Secure socket Layer)安全套接字层协议
  - 一种在客户端和服务器端之间建立安全通道的协议，主要是使用公开密钥体制和X.509数字证书技术保护信息传输的机密性和完整性
  - 它不能保证信息的不可抵赖性
  - 主要适用于点对点之间的信息传输，常用Web Server方式
- SSL包括：
  - 服务器认证
  - 客户认证（可选）
  - SSL链路上的数据完整性和SSL 链路上的数据保密性

SSL的作用

- 窃听风险（eavesdropping）
  - 第三方可以获知通信内容

所有信息都是加密传播，第三方无法窃听

- 篡改风险（tampering）
  - 第三方可以修改通信内容

具有校验机制，一旦被篡改，通信双方会立刻发现

- 冒充风险（pretending）
  - 第三方可以冒充他人身份参与通信

配备身份证书，防止身份被冒充

主要功能

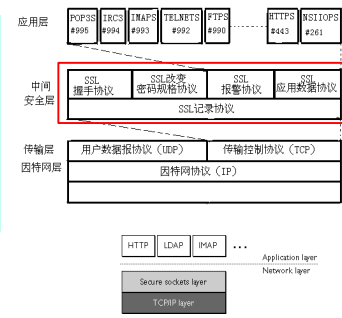
- SSL服务器认证
  - 允许用户确认服务器身份
  - 支持SSL协议的客户机软件能使用公钥密码标准技术检查服务器证书、公用ID是否有效和是否由在客户信任的CA列表内的认证机构发放
- SSL客户机认证（可选）
  - 允许服务器确认用户身份
  - 使用应用于服务器认证同样的技术，支持SSL协议的服务器软件能检查客户证书、公用ID是否有效和是否由在服务器信任的认证机构列表内的CA发放

主要功能（续）

- 机密性
  - 一个加密的SSL连接要求所有在客户机与服务器之间发送的信息由发送方软件加密和由接收方软件解密，这样提供了高度机密性
- 完整性
  - 所有通过加密SSL连接发送的数据都被一种检测篡改的机制所保护，这种机制自动地决定传输中的数据是否已经被更改

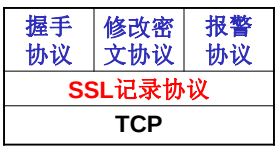
SSL/TLS协议栈

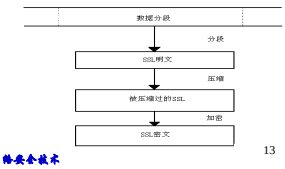
- 介于应用层和传输层之间
  - TCP层之上
  - 为应用层提供安全性服务



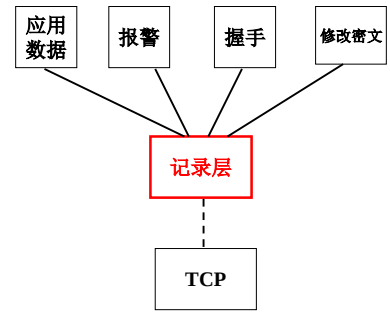
SSL协议的体系结构

- SSL协议工作过程
  - 在应用层通信之前就已经完成加密算法、通信密钥的协商以及服务器认证工作
  - 在此之后,应用层协议所传送的数据都被加密
- SSL是一个两层协议，包括：
  - 底层：SSL记录协议
  - 高层：
    - SSL握手协议
    - SSL修改密文协议
    - SSL警告协议





各种消息协同工作

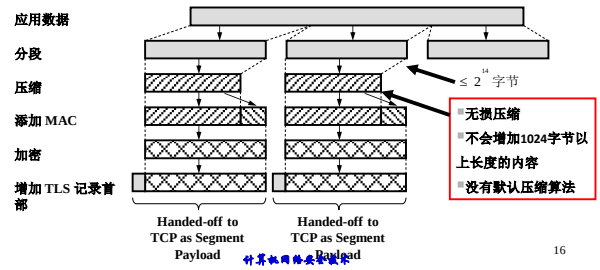


SSL记录协议

- 提供两种服务
  - 机密性：定义用于SSL数据加密的共享密钥
  - 消息完整性：定义用于产生MAC码的共享密钥
- SSL记录协议接收传输的应用报文
  - 将数据分片成可管理的块 **≤2^{14}-1=16383个字节**
  - 进行数据压缩(可选)
  - 应用消息认证码MAC
  - 接着利用IDEA、DES、3DES或其他加密算法进行数据加密,最后增加由内容类型、主要版本、次要版本和压缩长度组成的首部

SSL记录协议操作过程

- 把输入的任意长度的数据输出为统一标准的一系列的SSL数据段（或者叫“SSL记录”），每个这样的段最大为**2^{14}-1=16383个字节**。从原始数据段到生成SSL明文分段

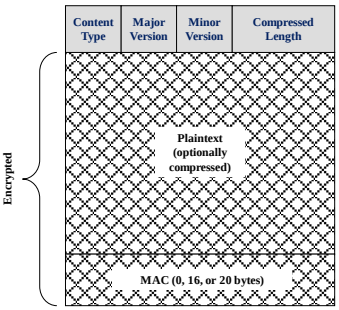


SSL记录协议密码套件

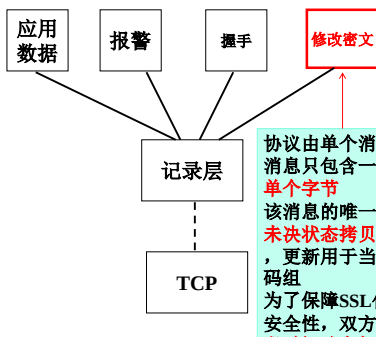
	密码套件
高强密码	3DES(192加密强度)-SHA1 AES256-SHA1
强密码	RC4(128)-MD5　AES(128)-MD5 RC2(128)-MD5　DES(56)-SHA1
出口密码	RC4(40)-MD5 RC2(40)-MD5
弱密码	NONE-MD5

SSL记录协议封装

- 内容类型（Content Type）(8 bits): 高层协议类型，
  包括修改密码规范，警报，握手，应用数据 4种
- 主要版本（Major Version）（8 bits），如SSLv3.0，为3
- 次要版本（Minor Version）（8 bits），如SSLv3.0, 为0
- 长度（Compressed length）(16 bits)：加密后数据的长度，不超过2^{14}+2048字节
- 密文数据（ EncryptedData fragment ）

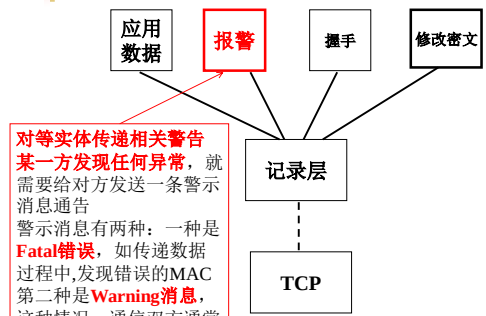


各种消息协同工作



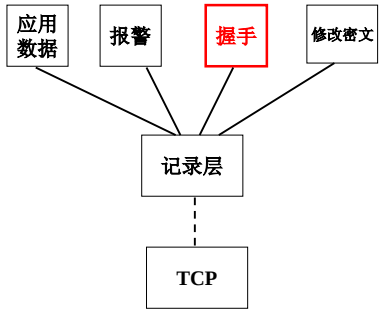
协议由单个消息组成,该消息只包含一个值为1的单个字节
该消息的唯一作用就是将未决状态拷贝为当前状态，更新用于当前连接的密码组
为了保障SSL传输过程的安全性，双方应该每隔一段时间改变加密规范

各种消息协同工作



对等实体传递相关警告
某一方发现任何异常，就需要给对方发送一条警示消息通告
警示消息有两种：一种是Fatal错误，如传递数据过程中,发现错误的MAC
第二种是Warning消息，这种情况，通信双方通常都只是记录日志

各种消息协同工作



SSL/TLS 会话 和 连接

◆**SSL Session**（会话）
- ❖指客户和服务器之间的一个关联关系
- ❖通过SSL握手协议创建session，它**确定了一组密码算法的参数**
- ❖Session可以**被多个连接共享**，从而可以避免为每个连接协商新的安全参数而带来 昂贵的开销
- ❖SSL Session都有一个当前状态

◆**SSL connection**（连接）
- ❖与**底层协议的端到端**相关联
- ❖**每个connection都与一个session**相关联
- ❖连接是**短暂的**

SSL会话状态

◆**实际上是一组参数**
- ❖**Session identifier**（会话号），字节序列，由服务器产生，用来标识一个会话状态
- ❖**Peer certificate**(可以为NULL)，对方的X.509 v3证书
- ❖**Compression method**，压缩数据的算法
- ❖**Cipher spec**，指定数据加密算法和用于HMAC的散列算法，以及算法的有关参数
- ❖**Master secret**, 客户和服务器之间**共享的48字节的会话密钥（主密钥）**
- ❖**Is resumable**，可恢复性，标记是否这个会话可以被用来初始化一个新的连接

SSL连接的状态

◆**连接状态也包含一组参数**
- ❖**Server and client random**，客户和服务器为每个连接选择的字节序列
- ❖**Server write MAC secret**，服务器在发送数据的时候，用于MAC运算的key
- ❖**Client write MAC secret**，客户在发送数据的时候，用于MAC运算的key
- ❖**Server write key**，服务器加密数据的密钥，以及客户解密数据的密钥
- ❖**Client write key**，客户加密数据的密钥，以及服务器解密数据的密钥
- ❖**Initialization vectors**，在CBC模式中用到的IV，最初由握手协议初始化，以后，每一个记录的最后一个密文块被用作下一个记录的IV
- ❖**Sequence numbers**，每一个连接都需要维护一个序列号，当密码参数变化时，重置为0

SSL握手协议
SSL Handshake Protocol

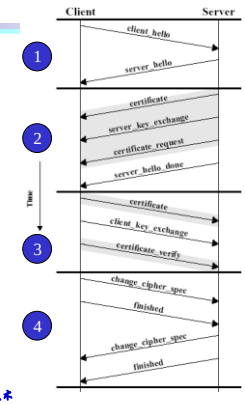
- 功能：客户端和服务器端相互认证，协商加密和MAC算法，密钥传送
- 位于SSL记录协议之上
  - 也用到了SSL记录协议的处理过程
  - ContentType = 22
  - 协议格式
    - 类型，长度，内容

Handshake Protocol

  - 当SSL客户和服务器开始通讯的时候，它们要通过协商，在以下信息方面获得一致:
    协议版本、密码算法、是否认证对方、
    用什么技术来产生共享秘密数据，等等

SSL握手协议的流程：四个阶段

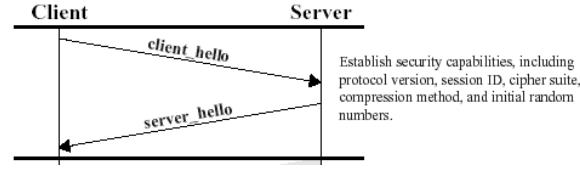
- 交换Hello消息，对于算法、交换随机值等协商一致
- 交换必要的密码参数，以便双方得到统一的premaster secret
- 交换证书和相应的密码信息，以便进行身份认证
- 产生master secret
- 把安全参数提供给SSL记录层
- 检验双方是否已经获得同样的安全参数



SSL握手协议使用的消息

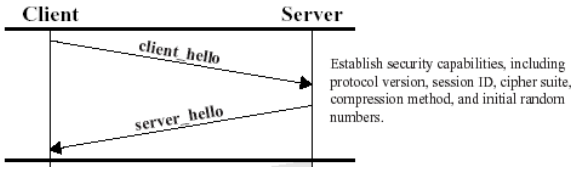
消息	参数
hello_request	Null
client_hello	版本，随机数，会话id，密码参数，压缩方法
server_hello	
certificate	X.509 v3证书链
server_key_exchange	参数，签名
certificate_request	类型，CAs
server_done	Null
certificate_verify	签名
client_key_exchange	参数，签名
finished	Hash值

第一阶段：建立安全能力属性（1）



- 客户发送一个client_hello消息
  - 1.版本
  - 2.随机数(32位时间戳+28字节随机序列) -防止重放攻击
  - **3.会话ID**
  - 4.客户支持的密码算法列表(CipherSuite)
  - 5.客户支持的压缩方法列表
- 然后，客户等待服务器的server_hello消息

第一阶段：建立安全能力属性（2）



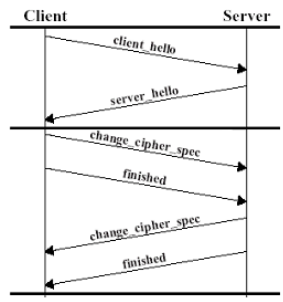
- 服务器发送server_hello消息
  - 1.客户机和服务器支持的最高SSL版本中的较低版本
  - 2.服务器产生的随机数
  - **3.会话ID**
  - 4.服务器从客户建议的密码算法中挑出一套
  - 5.服务器从客户建议的压缩方法中挑出一个

关于会话ID(Session ID)的讨论

- 客户方
  - **客户指定的会话ID如果不等于0**，则表示它希望基于**已有的这个会话**来更新已有连接的安全参数，或者创建一个新的连接
  - **如果会话ID等于0**，则表示客户希望**在一个新的会话上建立一个新的连接**
- 服务器
  - **同意客户指定的会话ID**，需要检查cache中的会话状态
  - **或者返回一个新的会话ID**

重用一个SSL会话

- 客户和服务器在交换hello消息中，客户要求重用已有的SSL会话，服务器同意使用cache中的已有的会话session id
- 跳过第二、第三阶段，直接把SSL会话中的参数传递给SSL记录层



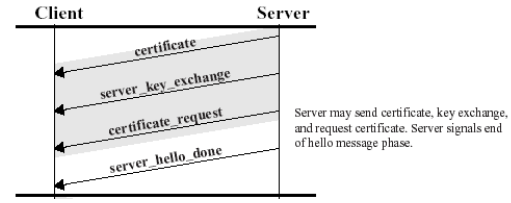
加密CipherSuite

- 第一个元素指定了密钥交换的方法，SSL支持以下一些方法：
  - **RSA，要求服务器提供一个RSA证书**
  - DH(Diffie-Hellman)，要求服务器的证书中包含了由CA签名的DH公钥参数。客户或者在证书中提供DH公钥参数，或者在密钥交换消息中提供此参数
  - EDH(Ephemeral Diffie-Hellman)，产生临时的密钥，DH公开参数由发送者的私钥进行签名，接收者用对应的公钥进行验证
  - 匿名的DH，不加认证。会受到中间人攻击

加密CipherSuite

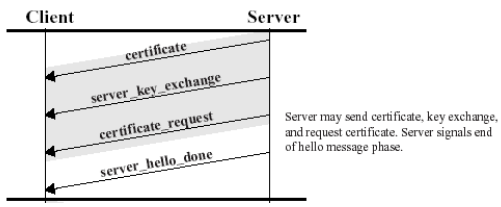
- 然后，指定以下信息
  - 加密算法，和类型(流还是分组密码算法)
  - HMAC算法，MD5还是SHA-1
  - HashSize
  - Key Material
  - IV Size

第二阶段：服务器认证和密钥交换（1）



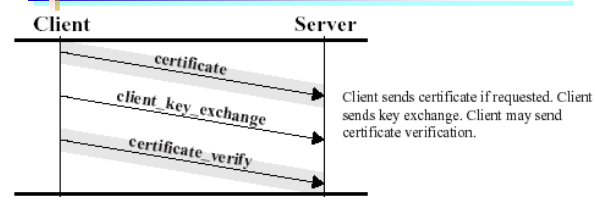
- 服务器发送自己的证书，消息包含**一个X.509证书**，或者**一条证书链**
  - 除了匿名DH之外的密钥交换方法都需要
- 服务器发送server_key_exchange消息
  - **可选的**，有些情况下可以不需要。只有当服务器没有向客户发送证书时，才发送此消息
  - 向客户发送公钥**（因为没有数字证书）**

第二阶段：服务器认证和密钥交换（2）



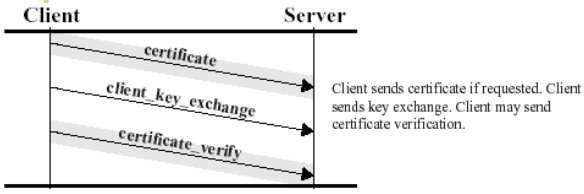
- 服务器发送certificate_request消息
  - **可选的**，非匿名server可以向客户请求一个证书
  - 包含证书类型和CAs
- 服务器发送server_hello_done, 然后等待应答

第三阶段：客户认证和密钥交换（1）



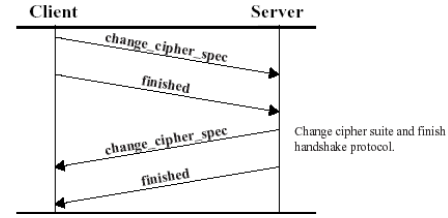
- 客户收到server_done消息后，它根据需要**检查服务器提供的证书**，并判断server_hello的参数是否可以接受，如果都没有问题的话，发送一个或多个消息给服务器
- **可选的**，如果服务器请求证书的话，则客户首先发送一个certificate消息，若客户没有证书，则发送一个no_certificate警告

第三阶段：客户认证和密钥交换（1）



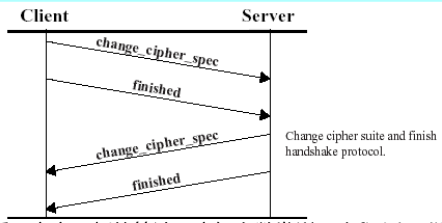
- 然后，**客户机密钥交换，客户机生成48位的预备主密码premaster secret**，用服务器公钥加密，发送给服务器
- 最后，客户发送一个certificate_verify消息，其中包含一个签名，对从第一条消息以来的所有握手消息的HMAC值(**用master_secret**)进行签名

第四阶段：结束（1）



- 第四阶段建立起一个安全的连接
- 客户发送一个change_cipher_spec消息，并且把协商得到的CipherSuite拷贝到当前连接的状态之中

第四阶段：结束（2）



- 然后，客户用新的算法、密钥参数发送一个finished消息，这条消息可以检查密钥交换和认证过程是否已经成功。其中包括一个校验值，对所有以来的消息进行校验
- 服务器同样发送change_cipher_spec消息和finished消息
- 握手过程完成，客户和服务器可以交换应用层数据

SSL性能分析

- SSL的应用**降低了**与HTTPS服务器和浏览器相互作用的速度
- **原因**：浏览器和服务器之间用来**初始化SSL会话和连接的状态信息需要用公钥加密和解密方案**
- 在开始连接到HTTPS服务器和收到**第一个HTML页时，用户经历了一个额外的几秒钟的停顿**
  - 因为SSL被设计成缓存以后会话中的主密钥
- 只影响浏览器和服务器之间的**第一次SSL连接**
  - 与创建会话相比，采用DES、RC2、或RC4算法来进行加密和解密数据的额外负担不算什么
- 对于高速计算机，相对联网速度而言，SSL会话或多个利用共享的主密钥的SSL会话建立后，传送大量数据时SSL协议的开销就显得微不足道

TLS和SSL的细微差异

- 版本号
  - 记录 TLS 主版本号 3，副版本号1
- 消息认证码
  - TLS使用RFC2104中的 HMAC算法（异或）
- 伪随机函数PRF
  - a pseudo-random function expands secrets
  - based on HMAC using SHA-1 or MD5
- 增加报警码
- 密码构件的差异
- 证书类型：changes in certificate types & negotiations
- 密码计算与填充：changes in crypto computations & padding

总结：SSL的安全性

- SSL提供了什么?
  - **SSL提供了通道级别的安全**: 连接的两端知道所传输的数据是保密的,而且没有被篡改
  - 几乎总是要**对服务器进行认证**
  - 可选的**客户端认证**
  - 针对异常情况的**安全通知**
    - 错误警示
    - 关闭连接
  - 所有这些**依赖于某些对系统的假定**
    - 假定已经正确产生了密钥数据
    - 该密钥已被安全地保管

总结：SSL的安全性(续)

- **保护master_secret**
  - 几乎协议的所有安全都依赖于master_secret的保密
  - 在内存中保护密钥
- **保护服务器的私有密钥**
  - 最常被违反的规则，很难保证做到
  - 要求**安全地存储私有密钥**
    - 多数实现都对磁盘上的私有密钥进行加密，而且提供口令保护
    - 其他实现在受保护的硬件中存储密钥
  - 上述两种方案在启动服务器时都要求管理员的介入，使得在系统崩溃或电力故障恢复时无法实现无人看管的重新启动
- **使用良好的随机数生成器**

针对SSL/TLS的攻击（1）

针对SSL/TLS的攻击（2）

- **一些常见的攻击手法**
  - **针对密钥算法的破解**
    - 取决于算法的强度，协商过程
  - **利用明文模式的攻击**
    - 上层协议中常常有一些固定的模式可以参考，比如http协议中get字节串
    - 构造字典(密文-密钥对)，查字典
    - TLS办法：用长密钥，使得不可能构造这样的字典
  - **重放攻击**
    - TLS中的nonce有32字节(包含时间戳)，可用于避免重放攻击
    - 会话ID标识了一个完整的会话，要重放部分会话需要知道私钥
  - **中间人攻击**
    - 通过证书来认证对方
    - 对于双方都是匿名的模式，中间人攻击也是成立的

OpenSSL

- OpenSSL创立于1998年，是一个支持SSL认证的服务器.它是一个开源软件，支持多种操作系统
  - 实现一个完整的、健壮的、商业级的开放源码工具，通过强大的加密算法来实现建立在传输层之上的安全性
  - OpenSSL包含一套SSL协议的完整接口，应用程序应用它们可以很方便的建立起安全套接层，进而能够通过网络进行安全的数据传输
  - 可为网络通信提供安全及数据完整性的一种安全协议
- 多数SSL加密网站使用OpenSSL开源软件包
  - 如网银、在线支付、电商网站、门户网站、电子邮件等
  - OpenSSH使用OpenSSL加密SSH
  - 是Apache Web服务器的默认安全通讯选项
  - 虚拟专用网络（VPN）

OpenSSL的特征

- 美国联邦政府NIST FIPS 140-2一级评估确认
- TLS，下一代SSL协议
- X.509密钥和证书的生成
- X.509证书权力
- S/MIME加密
- 文件加密和粉碎
- 打乱UNIX密码
- 9个不同的商业密码硬件设备
- 密码性能测试
- 36个命令
- 6个消息摘要算法
- 9个密码算法
- 多个加密协议

SSL的应用

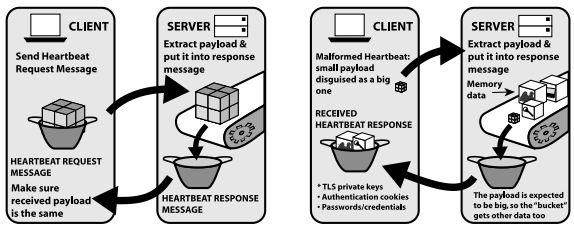
- SSL的典型应用主要有体现在两个方面，
  - **客户端**，主流浏览器（如IE和Netscape等）
  - **服务器端**，Web服务器（如**IIS、Apache、**Domino Go WebServer、Netscape Enterprise Server等）
- 实现浏览器（或其他客户端应用）和Web服务器（或其他服务器）之间的安全SSL信息传输
  - 在Web服务器端安装支持SSL的Web服务器证书
  - 在浏览器端安装支持SSL的客户端证书（可选）
  - 然后把URL中的“http://”更换为“https://”

“心脏出血”漏洞（Heartbleed）

补充

“Heartbleed”漏洞

- 2014年4月7日，OpenSSL宣布在OpenSSL 1.0.2-beta及1.0.1系列（2012年3月14日发布，除1.0.1g）的所有版本中，其所实现的TLS心跳扩展存在严重的内存处理错误.
- “心脏出血”漏洞Heartbleed
  - 是OpenSSL中的一个漏洞,自2011年12月31日，漏洞就已经存在
- 危害
  - 黑客可以对使用https(存在此漏洞)的网站发起攻击，每次读取服务器内存中64K数据，不断的反复获取，内存中可能会含有用户http原始请求、用户cookie甚至明文帐号密码等。
  - 使用https的网站，包括微信、淘宝、网银、社交、门户等。



(a) How TLS Heartbeat Protocol works

(b) How TLS Heartbleed exploit works

OpenSSL “心脏出血”漏洞分析

- ssl/dl_both.c

```
int
dtls1_process_heartbeat(SSL *s)
   {
   unsigned char *p = &s->s3->rrec.data[0], *pl;
   unsigned short hbtype;
   unsigned int payload;
   unsigned int padding = 16; /* Use minimum padding */
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

```
typedef struct ssl3_record_st
   {
      int type;            /* type of record */
      unsigned int length;   /* How many bytes available */
      unsigned int off;      /* read/write offset into 'buf' */
      unsigned char *data;   /* pointer to the record data */
      unsigned char *input;   /* where the decode bytes are */
      unsigned char *comp;   /* only used with decompression - malloc()ed */
      unsigned long epoch;   /* epoch number, needed by DTLS1 */
      unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
   } SSL3_RECORD;
```

```
unsigned char *buffer, *bp;
int r;
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

如果用户并没有在心跳包中提供足够多的数据，如pl指向的数据实际上只有一个字节，那么memcpy会把这条SSLv3记录之后的数据都复制出来。

修复漏洞

第一行语句抛弃了长度为0的心跳包
第二步检查确保了心跳包足够长。

修复代码中最重要的一部分如下：

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
   return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
   return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```


修复：升级openssl版本

- 查询自己网站是否中招
  - 访问 在线检测网站：http://wangzhan.360.cn/heartbleed，http://filippo.io/Heartbleed/
  - 输入自己网站的ip
  - 看网站输出结果
- 修复
  - 升级openssl版本1.0.1g 版本
    1）到http://www.openssl.org/source/下载最新版本的 openssl-1.0.1g.tar.gz
    2）解压安装： ./config && make && make install
    3）配置环境
    mv /usr/bin/openssl /usr/bin/openssl.OFF
    mv /usr/include/openssl /usr/include/openssl.OFF
    ln –s /usr/local/ssl/bin/openssl /usr/bin/openssl
    ln –s /usr/local/ssl/include/openssl /usr/include/openssl
    4）查看openssl版本是否对：openssl version -a，如果出现OpenSSL 1.0.1g证明正确
    5）重启linux，运行reboot命令
    6）Done
  - 重新更换https证书

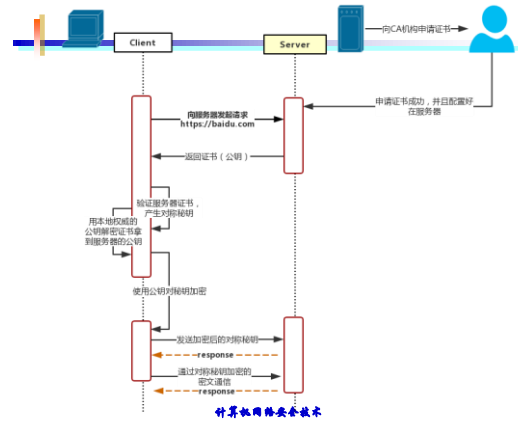
HTTPS

HTTPS

- HTTPS (HTTP over SSL)
  - https//：使用443端口
- HTTP与SSL/TLS相结合实现浏览器和WWW服务器之间的安全通信
  - RFC2818（HTTP over TLS）
- 加密以下信息
  - URL
  - document contents
  - form data
  - Cookies
  - HTTP headers

HTTPS的使用

- 建立连接
  - 客户端TLS握手(handshake)结束后， 发出第一个HTTP请求
    - **TLS ClientHello**
    - **http request**
- 关闭连接
  - HTTP记录中加入 “Connection: close”
  - TLS 关闭连接：使用TLS警告协议发送close_notify alerts
  - 关闭TCP连接
  - HTTP客户端还必须连接异常关闭的情况
    - TCP close before alert exchange sent or completed



说明

1. client向server发送请求https://baidu.com，然后连接到server的443端口。
2. 服务端必须要有一套公钥证书，可以自己制作，也可以向相关机构申请。
3. 传送公钥证书
4. 客户端解析证书：由客户端的TLS完成。首先验证公钥是否有效，如果证书没有问题，则生成一个随机值（秘钥）。然后用证书对该随机值进行加密。
5. 传送用证书加密后的秘钥：客户端与服务器端共享一个密钥
6. 服务端加密信息：服务端用私钥解密秘密秘钥，然后把内容通过该值进行对称加密。
7. 传输加密后的信息
8. 客户端解密信息：客户端用之前生成的私钥解密服务端传过来的信息，于是获取了解密后的内容。

针对https的攻击

1. HTTP 升级到 HTTPS
2. 针对证书的语义攻击
3. 无效证书
4. 混和内容
   - HTTP and HTTPS on the same page
5. 源污染（Origin contamination）
   - Weak HTTPS page contaminates stronger HTTPS page

SSH

（选）

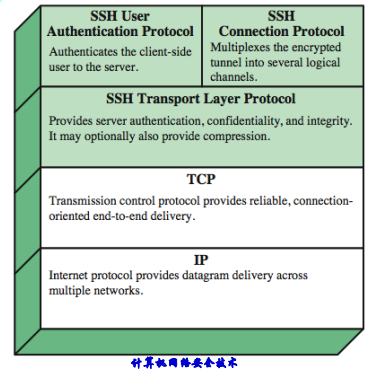
Secure Shell (SSH)

- SSH1 提供安全远程登录功能（secure remote logon ）
  - 代替 TELNET或其他不安全登录工具（如rlogin）
  - 增加其他客户/服务器功能
- SSH2 修复安全漏洞
  - RFCs 4250-4254
  - SSH 客户和服务器广泛应用于大多数操作系统中
  - 成为远程登录和X隧道的可选方法之一

SSH协议栈

- 传输层协议（Transport Layer Protocol）
  - 提供服务器认证、数据保密和带前向安全性的数据完整性（数据压缩可选）
- 用户认证协议（Authentication Methods）
  - 为服务器认证客户
- 连接协议（Connection Protocol）
  - 在单一的底层SSH连接上复用多个逻辑信道

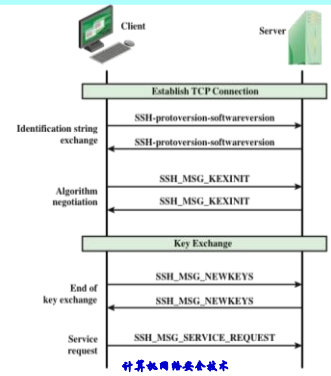
SSH 协议栈



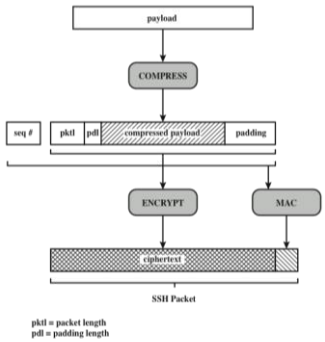
SSH 传输层协议

- 在传输层进行服务器认证，基于server/host 密钥对
  - 客户端需要预先知道服务器的公钥
- 分组交换
  - 建立TCP 连接
  - 交换数据
    - identification string exchange, algorithm negotiation, key exchange, end of key exchange, service request
  - 使用特殊包格式

SSH传输层包交换



SSH传输层包格式



SSH传输层密码算法

Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-MD5; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-MD5; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

* = Required
** = Recommended

SSH 用户认证协议

- 向服务器认证客户端
- 三种消息类型
  - SSH_MSG_USERAUTH_REQUEST
  - SSH_MSG_USERAUTH_FAILURE
  - SSH_MSG_USERAUTH_SUCCESS
- 认证方法
  - public-key, password, host-based

认证方法

- 公开密钥Publickey
  - 客户端向服务器发送的消息中包括：客户端的公钥，并用自己的私钥进行签名
  - 服务器收到后，检查公钥的有效性，如果可以就验证签名是否正确。
- 口令Password
  - 客户端发送的消息包括明文口令密码，并被传输层协议（Transport Layer Protocol）加密。
- 基于主机 Hostbased
  - 认证客户端主机，而非用户本身
  - 客户端发送一个用客户端主机的私有密钥进行签名的消息
  - SSH服务器验证客户端主机的签名，而非直接验证用户ID

SSH 连接协议

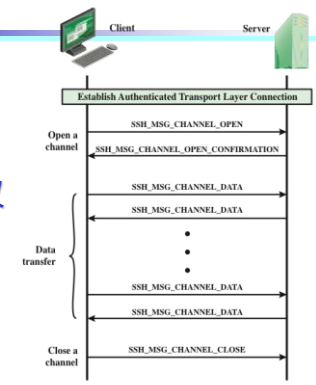
◆运行在SSH传输层协议之上

◆假设：一个安全身份认证连接正在使用中

◆连接协议使用多重逻辑信道

- ❖SSH 通信使用独立信道
- ❖通信双方都可以开启信道，每一端具有唯一的信道id
- ❖使用窗口机制进行流量控制
- ❖三个阶段:
  - ➢打开信道；数据传输；关闭信道
- ❖四种信道类型:
  - ➢session, x11, forwarded-tcpip, direct-tcpip.

SSH 连接协议



四种信道类型

会话Session

- The remote execution of a program
- The program may be a shell, an application such as file transfer or e-mail, a system command, or some built-in subsystem
- Once a session channel is opened, subsequent requests are used to start the remote program

X11

- Refers to the X Window System, a computer software system and network protocol that provides a graphical user interface (GUI) for networked computers
- X allows applications to run on a network server but to be displayed on a desktop machine

前向 Forwarded-tcpip

- Remote port forwarding

直接 Direct-tcpip

- Local port forwarding

端口转发（Port Forwarding）

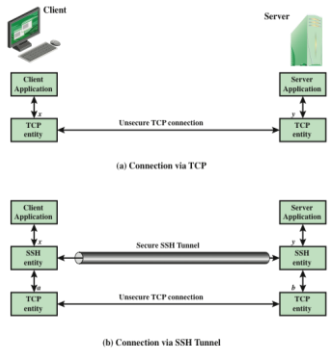
◆将不安全的TCP连接转换为安全的SSH连接（SSH的隧道技术）

- ❖SSH传输层协议在SSH客户端和服务器之间建立一个 TCP 连接
- ❖客户端流量被重定向到本地SSH，通过隧道传输到远程服务器

◆支持两种类型的端口映射

- ❖本地映射 – hijacks selected traffic
- ❖远程映射 – client acts for server

SSH传输层包交换



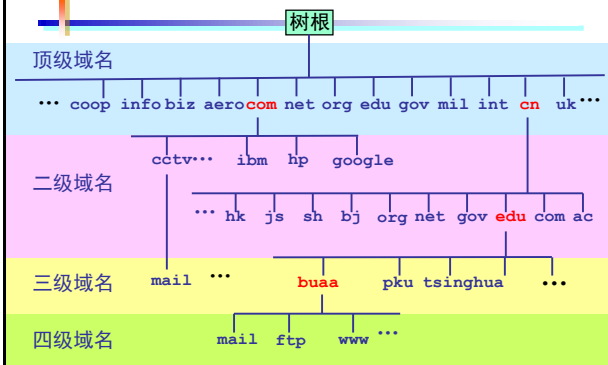
DNS安全

域名系统DNS

- 域名
  - 域名是用来标识网络上的主机，它与主机的IP地址相对应，一台主机可以有多个域名。
- 层次树状结构的命名方法
  - 任何一个连接在因特网上的主机或路由器，都有一个惟一的层次结构的名字，即域名。
  - 域名的结构由若干个分量组成，各分量之间用点隔开：

    … .三级域名.二级域名.顶级域名
- 各分量分别代表不同级别的域名，域名系统是分级的分布式数据库系统，用来查找域名与IP地址的对应关系。
- 查询域名的应用程序叫解析器（resolver），存储域名与IP地址对应关系的服务器叫域名服务器。
- DNS报文传输层可采用TCP或UDP协议，端口号均为53号

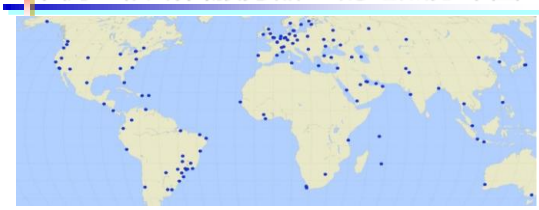
因特网的名字空间



根域名服务器共有 13 套装置

- 根域名服务器共有 13 套装置，不是 13 个机器。
- 这些根域名服务器相应的域名分别是:
  a.root-servers.net
  b.root-servers.net
  ...
  m.root-servers.net
- 到2016年2月，全世界已经在 588 个地点安装了根域名服务器，使世界上大部分 DNS 域名服务器都能就近找到一个根域名服务器。
- 大部分采用任播（Anycast）技术，编号相同的根服务器使用同一个IP。
  - 中国大陆在北京有三台编号为L的镜像，编号为F、I、J的镜像各一台，共6台；香港有编号为D、J的镜像各2台，编号为A、F、I、L的镜像各一台，共8台；台湾则有编号为F、I、J各一台，共3台

举例：根域名服务器 L 的地点分布图



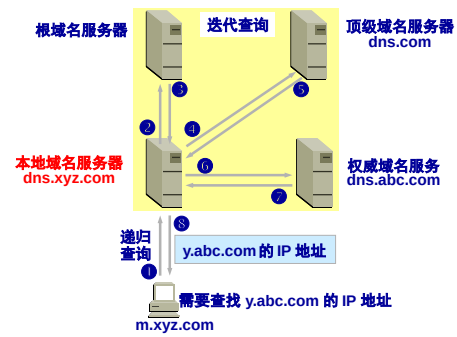
根域名服务器 L 分布在世界 150 个地点

- **根域名服务器并不直接把域名直接转换成 IP 地址。**
- **在使用迭代查询时，根域名服务器把下一步应当找的顶级域名服务器的 IP 地址告诉本地域名服务器。**

域名的解析过程

- 主机向本地域名服务器的查询通常采用递归查询。
  - 如果主机所询问的本地域名服务器不知道被查询域名的 IP 地址，那么本地域名服务器就以 DNS 客户的身份，向其他根域名服务器继续发出查询请求报文。
- 本地域名服务器向根域名服务器的查询通常采用迭代查询。
  - 当根域名服务器收到本地域名服务器的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地域名服务器：“你下一步应当向哪一个域名服务器进行查询”。然后让本地域名服务器进行后续的查询。

本地域名服务器采用迭代查询



名字的高速缓存

- 每个域名服务器都维护一个高速缓存，存放最近用过的名字以及从何处获得名字映射信息的记录。
  - 减轻根域名服务器的负荷，使互联网上的 DNS 查询请求和回答报文的数量大为减少。
- 为保持高速缓存中的内容正确，域名服务器应为每项内容设置计时器，并处理超过合理时间的项（例如，每个项目只存放两天）。
- 当权威域名服务器回答一个查询请求时，在响应中都指明绑定有效存在的时间值。
  - 增加此时间值可减少网络开销，而减少此时间值可提高域名转换的准确性。

DNS的安全性

- DNS响应被缓存
  - 提高重复解析的响应速度
  - 缓存信息还可以协助别的域名服务器解析，当其他客户端请求域名记录时，相匹配的缓存信息能够迅速回复新的查询请求。
- DNS缓存异常查询
  - 对于异常查询（如错误拼写）能迅速返回，节省响应时间
- 缓存数据的生命期
  - 数据的生命期TTL由域名数据的属主进行控制 生存时间(TTL)
  - TTL值适用于所有缓存的域名信息。最小的 TTL值是 1 小时，还可以进行默认时间调整，根据需要可以在每个 RR 上，灵活设置各自的缓存 TTL 值。

作业

- SSL协议由哪些协议构成?
- SSL连接和SSL会话之间有什么不同?
- 考虑Web的安全威胁，如中间人攻击，口令嗅探、IP劫持等，结合SSL的特征说明每个威胁是如何解决的?